Problem A. Adrenaline Rush

Time limit:	3 seconds
Memory limit:	1024 megabytes

Alice's friend is a big fan of the Adrenaline Rush racing competition and always strives to attend every race. However, this time, Alice is the one watching the race. To ensure her friend does not miss any important details, Alice decides to take notes on everything that happens on the track.

The first thing Alice notices before the race begins is the numbering of the cars. All the cars line up in front of the starting line in a specific order. The car closest to the line is numbered 1, the second car is numbered 2, and so on, up to the last car, which is numbered n. How convenient! — Alice thought.

The race begins with the countdown: "Three! Two! One! Go!". Alice observes that the cars start in their original order. However, as the race progresses, their order changes. She records whenever one car overtakes another, essentially swapping places with it on the track.

During the race, Alice notices something curious: no car overtakes another more than once. In other words, for any two cars x and y, there are at most two overtakes between them during the race: "x overtakes y" and/or "y overtakes x".

At the end of the race, Alice carefully writes down the final order of the cars $c_1, c_2, \ldots, c_n$, where c_1 represents the winner of the race.

Alice's friend, however, is only interested in the final ranking and discards all of Alice's notes except for the final ordering. As Alice is quite curious, she wonders: *What is the longest possible sequence of overtakes she could have observed during the race?* Your task is to help Alice answer this question.

Input

The first line of the input contains a single integer n ($1 \le n \le 1000$) — the number of cars in the race.

The second line contains a permutation $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le n, c_i \ne c_j$) — the final order of the cars.

Output

The first line of the output should contain a single integer m — the maximum possible number of overtakes that can occur during the race.

Each of the next m lines should contain two integers x and y ($1 \le x, y \le n$, $x \ne y$) representing an overtake event, where car x overtakes car y. This means that car x was directly behind car y and overtakes it. The overtakes must be listed in the order they occurred during the race.

After all m overtakes have occurred, the cars must arrive at the finish line in the order $c_1, c_2, \ldots, c_n$. Note that any car x should not overtake another car y more than once.

If there are multiple possible longest sequences of overtakes, output any of them.

Examples

standard input	standard output

3
2 3 1
	4
	2 1
	3 1
	3 2
	2 3

1 1	0

2
1 2
	2
	2 1
	1 2

Problem B. BitBitJump

Time limit:	3 seconds
Memory limit:	1024 megabytes

BitBitJump is a one instruction set computer. Thus, it has only one instruction: `bbj a, b, c`, which copies an a-th bit of memory to the b-th bit of memory and then jumps to address c.

Let's consider a 16-bit BitBitJump computer. It has 2^{16} bits of memory organized in 2^{12} 16-bit words. Words are counted from 0, and bits in a word are counted from the least significant (0-th) bit to the most significant (15-th) bit.

This computer has a single instruction pointer register (IP), and execution starts with IP $= 0$. If the current IP $\geq 2^{12} - 2$, the computer stops. Otherwise, it uses the IP-th word as a, the $(\text{IP} + 1)$-th word as b, the $(\text{IP} + 2)$-th word as c, and performs the `bbj a, b, c` instruction: copies the $(a \,\&\, 15)$-th bit of the $(a \gg 4)$-th word to the $(b \,\&\, 15)$-th bit of the $(b \gg 4)$-th word, and sets IP $= c$. Here, '&' represents bitwise AND, and '$\gg$' represents bitwise shift right operation. Notice that the value of c is read from memory after the bit copy, so if the instruction modified its own c, the new value will be used for IP.

For example, the `bbj 32, 35, 5` instruction placed at the memory start will be executed as follows:

1. $a = 32$ and $b = 35$ are read from the memory.
2. The 0-th bit of the 2-nd word (its value is $5 \,\&\, 1 = 1$) will be copied to the 3-rd bit of the same word, so the 2-nd word will have the value of $5 + 2^3 = 13$.
3. Then $c = 13$ is read from memory, and IP is set to 13.

Let's call the $(2^{12} - 1)$-th word ($2^{16} - 16 \ldots 2^{16} - 1$-th bits of memory) an *IO-word*. An *x-comparator* is a program which checks whether the value of the IO-word is equal to x. It should stop after execution of no more than 2^{12} instructions, leaving the lowest bit of the IO-word equal to 1 if the original value of the IO-word was equal to x, and 0 otherwise.

Write a program that generates an x-comparator for the given value of x.

Input

The input contains a single decimal integer x ($0 \leq x < 2^{16}$) — the value for which to build the x-comparator.

Output

The output should contain the x-comparator program dump. Dump consists of values for the first n words of the memory ($1 \leq n \leq 2^{12} - 1$). All other words, except the IO-word, are filled with zeroes.

For each of the n words, output its value as a four-character hexadecimal number. Values should be delimited by space or new line characters.

Example

standard input
```
0
```

standard output
```
fff0 0026 0003  fff1 0056 0006  fff2 0086 0009  fff3 00b6 000c  fff4 00e6 000f
fff5 0116 0012  fff6 0146 0015  fff7 0176 0018  fff8 01a6 001b  fff9 01d6 001e
fffa 0206 0021  fffb 0236 0024  fffc 0266 0027  fffd 0296 002a  fffe 02c6 002d
ffff 02f6 0030
0004 fff0 0fff
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 fff0 0fff  0000 fff0 0fff  0000 fff0 0fff  0000 fff0 0fff  0000 fff0 0fff
0000 fff0 0fff  0000 fff0 0fff  0000 fff0 0fff  0000 fff0 0fff  0000 fff0 0fff
0000 fff0 0fff  0000 fff0 0fff  0000 fff0 0fff  0000 fff0 0fff  0000 fff0 0fff
0000 fff0 0fff
```

Note

A dump in the sample output contains a 0-comparator. It consists of the following blocks:

- 16 instructions: the i-th of them, counting from 0, copies the i-th bit of the input word to the 6-th bit of its own c. If the copied bit is zero, it will proceed to the next instruction; otherwise, the next instruction number will be increased by 64.
- The following instruction copies the 4-th bit of the 0-th word (value 1) to the 0-th bit of the IO-word and jumps to the stop address.
- 16 unused words filled with 0.
- 16 equal instructions (starting from word 67). Each of them copies the 0-th bit of the 0-th word (value 0) to the 0-th bit of the IO-word and jumps to the stop address.

Problem C. Cactus without Bridges

Time limit:	3 seconds
Memory limit:	1024 megabytes

Caroline asked you for help in solving a cactus problem one year ago. During the last year, she researched extensively about cactuses. Today, she is the one presenting the problem.

You are given a **cactus without bridges** and also **the length of each odd simple cycle is greater than or equal to the number of odd simple cycles in cactus**. Your task is to answer whether it's possible to label the cactus edges with positive integers such that the following conditions are satisfied:

- Let's define the maximum label with t. All the integers $1, 2, \ldots, t$ are used in labeling (note that you do not need to minimize or maximize the value of t);
- For each vertex v of the given cactus, the labels of edges incident to the vertex v should be different and should form an interval of consecutive integers.

An edge in the graph is called *bridge* if the deletion of that edge increases the number of connected components of the graph. A *cactus* is a connected undirected graph in which every edge lies on at most one simple cycle. Intuitively, a cactus is a generalization of a tree where some cycles are allowed. Multiedges (multiple edges between a pair of vertices) and loops (edges that connect a vertex to itself) are not allowed in a cactus.

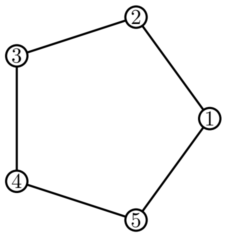
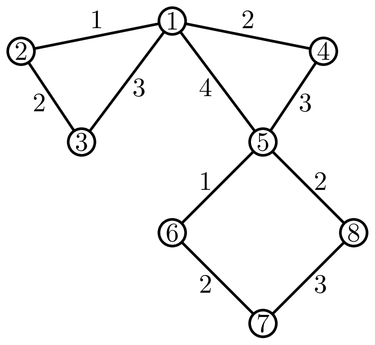
Input

The first line contains two integers n and m ($3 \le n \le 10^5$, $n \le m \le \lfloor \frac{3(n-1)}{2} \rfloor$) — the number of vertices and edges in the cactus. Each of the next m lines contains two integers u and v ($1 \le u, v \le n$; $u \ne v$) — the edges of the cactus. The given cactus satisfies all constraints from the problem statement.

Output

If finding the labeling satisfying the problem's conditions is impossible, output the single line with the word "NO". Otherwise, in the first line output the single word "YES". In the second line output an integer t ($1 \le t \le m$) — the number of different labels. In the third line output should contain m integers c_i ($1 \le i \le m$, $1 \le c_i \le t$) — the labels of the edges.

Examples

standard input	standard output	Illustration
5 5	NO	
1 2		
2 3		
3 4		
4 5		
5 1		
8 10	YES	
1 2	4	
2 3	1 2 3 2 4 3 1 2 3 2	
1 3		
1 4		
1 5		
4 5		
5 6		
6 7		
7 8		
8 5		

Problem D. DAG Serialization

Time limit:	3 seconds
Memory limit:	1024 megabytes

Consider a simple single-bit boolean register that supports two operations:

- **set** — sets the register to **true** if it was **false**, and returns **true**; otherwise, it returns **false**;
- **unset** — sets the register to **false** if it was **true**, and returns **true**; otherwise, it returns **false**.

The initial state of the register is **false**. Suppose there were n operations op_i (for $1 \le i \le n$) where **at most two operations returned true**. Also, we are given the partial order of operations as a directed acyclic graph (DAG): an edge $i \to j$ means that op_i happened before op_j. You are asked whether it is possible to put these operations in some linear sequential order that satisfies the given partial order and such that if operations are applied to the register in that order, their results are the same as given.

Input

In the first line, you are given an integer n — the number of operations ($1 \le n \le 10^5$). In the following n lines, you are given operations in the format "*type result*", where *type* is either "`set`" or "`unset`" and *result* is either "`true`" or "`false`". It is guaranteed that at most two operations have "`true`" results.

In the next line, you are given an integer m — the number of arcs of the DAG ($0 \le m \le 10^5$). In the following m lines, you are given arcs — pairs of integers a and b ($1 \le a, b \le n$; $a \ne b$). Each arc indicates that operation op_a happened before operation op_b.

Output

Print any linear order of operations that satisfies the DAG constraints and ensures the results of the operations match the ones given in the input. If a correct operation order does not exist, print -1.

Examples

standard input	standard output
5 set true unset true set false unset false unset false 2 1 4 5 2	5 1 3 2 4
3 unset true unset false set true 0	2 3 1
2 unset false set true 1 2 1	-1
2 unset false set false 0	-1

Problem E. Expression Correction

Time limit:	3 seconds
Memory limit:	1024 megabytes

Eve is studying mathematics in school. They've already learned how to perform addition and subtraction of decimal numbers and are practicing it by solving fun puzzles. The specific type of the puzzle they are solving is described below. They are given an equality with addition and subtraction which may or may not be a correct one. They have to verify the equality, and if it is not a correct one, then they have to tell if it is possible to turn it into a correct one by moving one digit to a different place in the equality.

Let us formally define the equality in this puzzle:

- *Number* is a string of **at least one and at most 10 decimal digits** ('`0`' to '`9`') that has no extra leading zeroes (the only number that is allowed to start with the zero digit is "`0`").
- *Expression* is a string composed of one or more numbers, as defined above, that are separated with addition ('`+`') or subtraction ('`-`') operators.
- *Equality* is a string composed of an expression, as defined above, followed by an equals sign ('`=`'), followed by another expression.
- *Correct equality* is an equality where both expressions on the left and right hand sides of the equals sign evaluate to the same decimal number according to the standard arithmetic. Note that while all the numbers in the expression are positive, the evaluated number can be negative. Also, the evaluated number can be longer than 10 digits.
- *Moving a digit* in an equality means removing a digit from any position in the string and inserting it into another position so that the resulting string is again an equality.

The puzzle is pretty straightforward once you know how to add and subtract decimal numbers, but it is tenuous. It is easy to get distracted and make a mistake while performing computation. Your task is to write a program that solves the expression correction puzzle to help Eve.

Input

The input file consists of a single line — an equality as defined in the problem statement. The total length of the input string does not exceed 100 characters.

Output

Write a single line to the output. If the input contains a correct equality, output a single word "`Correct`". Otherwise, if the input equality can be turned into a correct one by moving one digit, output the resulting correct equality. If there are multiple possible correct equalities after moving one digit, you may output any one of them. Otherwise, output a single word "`Impossible`".

Examples

standard input	standard output
`2+2=4`	`Correct`
`123456789+9876543210=111111110+11-1`	`123456789+987654321=1111111100+11-1`
`10+9=10`	`Impossible`
`24=55-13`	`42=55-13`
`1000000000-10=9999999999`	`Impossible`

Problem F. Fix Flooded Floor

Time limit:	3 seconds
Memory limit:	1024 megabytes

Archimedes conducted his famous experiments on buoyancy. But not everyone knows that while he was taking a bath, he was too focused and didn't notice the moment when the water overflowed over the edge of the bath and flooded the floor near the wall. His expensive parquet was irreversibly damaged!

Archimedes noticed that not all was lost, and there were still several undamaged parquet pieces. The parquet near the wall had the shape of a long narrow stripe of $2 \times n$ cells. Archimedes had an unlimited supply of 1×2 parquet pieces that could be placed parallel or perpendicular to the wall. Archimedes didn't want to cut the parquet pieces. As a great scientist, he figured out that there was exactly one way to restore the parquet by filling the damaged area of the parquet with the non-overlapping 1×2 cell shaped pieces.

Help historians to check Archimedes' calculations. For the given configuration of the $2 \times n$ parquet floor, determine whether there is exactly one way to fill the damaged parquet cells with the 1×2 cell parquet pieces. If Archimedes was wrong, find out whether there are multiple ways to restore the parquet, or there are no ways at all.

Input

The first line contains a single integer T ($1 \le T \le 10^4$) — the number of test cases to solve.

Then the description of test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the parquet floor.

The following two lines contain exactly n characters each and describe the parquet, where '`.`' denotes a damaged cell and '`#`' denotes an undamaged cell.

The total sum of n in all T test cases doesn't exceed $2 \cdot 10^5$.

Output

For each test case, print "`Unique`" if there is exactly one way to restore the parquet, "`Multiple`" if there are multiple ways to do so, or "`None`" if it is impossible to restore the parquet.

Example

standard input	standard output
4 10 #.......## ##..#.##.. 6 ...#.. ..#... 8 .... 3 ### ###	Unique None Multiple Unique

Problem G. Geometric Balance

Time limit: 3 seconds
Memory limit: 1024 megabytes

Peter's little brother Ivan likes to play with a turtle. The turtle is a special toy that lives on the plane and can execute three commands:

- Rotate a degrees counterclockwise.
- Draw d units in the direction it is facing while dispensing ink. No segment of the plane will be covered by ink more than once.
- Move d units in the direction it is facing without drawing.

Ivan just learned about the compass, so he will only rotate his turtle so it faces one of eight cardinal or ordinal directions (angles a in rotate commands are always divisible by 45). Also, he will perform at least one draw command.

Peter has noted all the commands Ivan has given to his turtle. He thinks that the image drawn by the turtle is adorable. Now Peter wonders about the smallest positive angle b such that he can perform the following operations: move the turtle to a point of his choosing, rotate it by b degrees, and execute all the commands in the same order. These operations should produce the same image as the original one. Can you help Peter?

Note, two images are considered *the same* if the sets of points covered by ink on the plane are the same in both of the images.

Input

The first line of the input contains a single integer n ($1 \le n \le 50000$) — the number of commands Ivan has given.

The next n lines contain commands. Each command is one of:

- "`rotate` a" ($45 \le a \le 360$) where a is divisible by 45;
- "`draw` d" ($1 \le d \le 10^9$);
- "`move` d" ($1 \le d \le 10^9$).

At least one and **at most 2000** of the commands are `draw`. It is guaranteed that no segment of the plane will be covered by ink more than once.

Output

Output a single number, the answer to the question. The answer always exists.

Examples

standard input	standard output

`1`
`draw 10`
	`180`

`7`
`draw 1`
`rotate 90`
`draw 1`
`rotate 90`
`draw 1`
`rotate 90`
`draw 1`
	`90`

`3`
`draw 1`
`move 1`
`draw 2`
	`360`

Problem H. Hunting Hoglins in Hogwarts

Time limit:	6 seconds
Memory limit:	1024 megabytes

This is an interactive problem.

Harry and Hermione are trying to hunt down *hoglins* which are haunting Hogwarts. There is a long hallway in Hogwarts, consisting of n individual *cells*, numbered from 1 to n from the left to the right.

Hermione can cast a spell that would *block* any cell of the hallway of her choosing. After the spell is cast, the blocked cell will remain blocked while she casts other spells.

Hoglins are simple creatures; all they do is randomly move around and bump into stuff. To be more precise, every hoglin has a range which it considers to be *accessible*. Initially, when the hoglin appears, it is a range from the cell 1 to the cell n.

Initially, a single hoglin appears in a cell of the hallway chosen uniformly at random. Then, until this hoglin is caught, the following happens on every *round* of the hunt:

- Hermione can cast a spell to block any single cell of her choosing, or do nothing.
- If the cell she is trying to block is the cell with a hoglin in it, the hoglin is caught. After that, all the blocked cells become free again, and, if there are more hoglins to be caught, a new hoglin immediately appears in a random location, and the hunt begins again.
- Otherwise, the hoglin chooses a cell uniformly at random from its accessible range and tries to move to that cell, moving one cell at a time towards a chosen cell. Regardless of the distance, all the steps of the movement, as described below, happen in the same round.
- If the chosen cell is to the right of the hoglin, it moves to the right; if the chosen cell is to the left of the hoglin, it moves to the left. If the chosen cell is the same as where the hoglin is now, it does nothing.
- If at any point during the movement towards the chosen cell a hoglin is trying to move to the right or to the left from an unblocked cell at position i to the neighbouring blocked cell at position $i \pm 1$, the hoglin updates the right or left boundary of its accessible range correspondingly to be i.
- If on the way to the chosen cell, the hoglin tries to move to a blocked cell, Harry and Hermione hear a loud sound, as the hoglin *bumps* into the blocked cell. In this case, the hoglin returns to the position it has originally started from at the beginning of this round.
- Otherwise, if the hoglin does not bump into any blocked cells on its way, it does not change its accessible range and stays at the new position. In that case, Harry and Hermione hear nothing.

To free Hogwarts from hoglins, Harry and Hermione should catch k of them, but they don't have much time. They can only afford to hunt hoglins for at most 200 000 rounds. Please help them find an efficient strategy to do that.

Interaction Protocol

First, the testing system will write two integers n and k ($1 \le n \le 10^{18}; 1 \le k \le 800$) — the number of cells in Hogwarts' hallway and the number of hoglins that should be caught. Then the catching process begins.

The following interaction proceeds in rounds as described in the problem statement.

At the start of each round, your program should output Hermione's action — an integer p ($0 \le p \le n$) representing the position of the cell Hermione is going to block. If $p = 0$ or the cell at position p is already blocked, she does nothing in this round.

Then, if the current position of the hoglin is at the newly blocked cell p, the hoglin is caught; the testing system outputs `2`, all the blocked cells become free, and interaction rounds start again. In case you caught the k-th hoglin, the testing system outputs `3` instead of `2`, and your program should immediately stop execution.

Otherwise, the hoglin attempts to move according to the rules described in the problem statement. If in the process it bumps into any blocked cell, the testing system outputs `1`; otherwise, it outputs `0`.

If your 200 000-th action does not catch the k-th hoglin, the testing system outputs `-1` instead of its usual answer, and your program should immediately stop execution to guarantee the "Wrong Answer" verdict.

The interactor in this problem is not adaptive. It is guaranteed that the hoglins follow the rules described in the problem statement. The starting cell for each hoglin is chosen uniformly at random and their moves are chosen uniformly at random from the range of cells that they consider accessible.

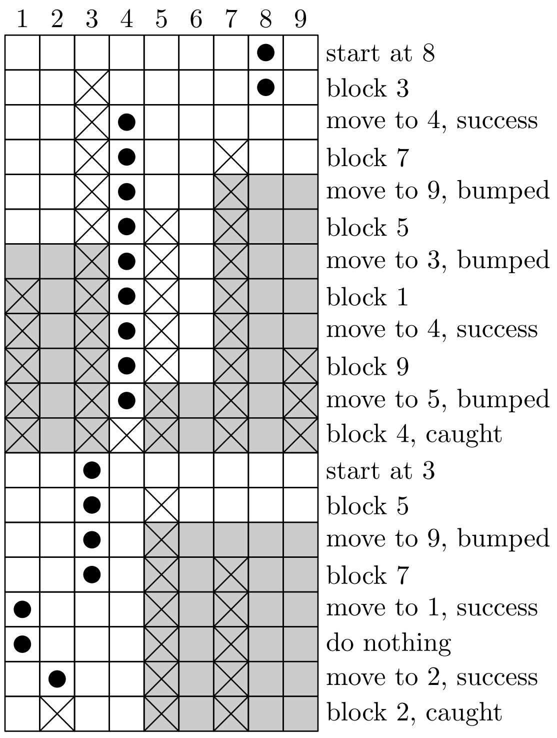
The problem has at most 15 tests.

Here is the summary of all possible interactor answers:

- `-1` — too many actions;
- `0` — hoglin moved successfully, did not bump;
- `1` — hoglin attempted to move, bumped into blocked cell;
- `2` — hoglin is caught, interaction starts again;
- `3` — hoglin is caught, stop.

Example

standard input	standard output	Illustration

9 2
0
1
1
0
1
2
1
0
0
3
	3
	7
	5
	1
	9
	4
	5
	7
	0
	2
		 1 2 3 4 5 6 7 8 9
		start at 8
		block 3
		move to 4, success
		block 7
		move to 9, bumped
		block 5
		move to 3, bumped
		block 1
		move to 4, success
		block 9
		move to 5, bumped
		block 4, caught
		start at 3
		block 5
		move to 9, bumped
		block 7
		move to 1, success
		do nothing
		move to 2, success
		block 2, caught

Note

We show the sample from the point of view of the hoglins.

The black dot shows the current position of the hoglin.

Crosses mark blocked cells.

White cells mark the range which the hoglin considers to be accessible; other cells are marked gray.

On the right is the action that was performed by either Hermione or the hoglin to get to this state from the previous one.

Problem I. Incompetent Delivery Guy

Time limit:	3 seconds
Memory limit:	1024 megabytes

In Isengard, wizard Saruman, with the help of some magic spells, organized a transport system between n towers. To be precise, he created m one-directional passages, each connecting two towers. Each passage i has some number t_i associated with it, meaning the time it takes, in seconds, for an orc to travel along it. In other words, Saruman's transport system can be represented with a directed weighted graph.

On December 15th, Saruman, sitting in the middle tower called *Orthanc*, gets a message from Sauron (via palantir) which says that a valuable present is already near Isengard's entrance tower. So, Saruman needs to instruct the garrison to select one of the orcs and send them with a gift along the shortest path from the entrance tower to Orthanc.

Unfortunately, orcs... aren't exactly smart fellas. Although they are able to drag a load along passages in the transport system and they (at least, in principle) know where Orthanc is, orcs have a really poor understanding of the concept of the shortest path. To make everyone's lives easier, on some towers Saruman puts a huge flashing pointer which says "**TO ORTHANC — THIS WAY**" and points to one of the passages leading from this tower. Saruman wants orcs to reach Orthanc as fast as possible — hence, a flashing pointer can only point to a passage that lies on one of the shortest paths to Orthanc. Formally, the flashing pointer on the tower u can point to a passage $\vec{a} = \overrightarrow{uv}$ only if $\overrightarrow{\mathrm{dist}}(v, O) < +\infty$ and $\overrightarrow{\mathrm{dist}}(u, O) = t_{\vec{a}} + \overrightarrow{\mathrm{dist}}(v, O)$. Here by $\overrightarrow{\mathrm{dist}}(x, y)$ we denote the minimum time it takes to get to tower y from tower x (or $+\infty$ if there is no oriented path from x to y), by O we denote the tower of Orthanc, and by u and v we denote the starting and the finishing towers of the passage $\vec{a}$. Note that Saruman will not put a flashing pointer onto Orthanc, nor will he put it on any tower from which Orthanc is unreachable. On each of the remaining towers, he will put exactly one flashing pointer.

This still does not work perfectly well. While traveling to Orthanc, each time an orc is near some tower (any but Orthanc), the orc can either choose a marked passage with Saruman's sign or do some *hanging around*, as Saruman calls it, when the orc chooses an outgoing passage completely at random. For any orc, there exists an integer d such that when they're given an order to go to Orthanc, during the commute the orc never chooses to hang around more than d times. This exact number we will politely call this orc's *incompetence.*

Note that at some moment it may happen that an orc finds themselves near a tower from which there is no oriented path to Orthanc. Under these unfortunate circumstances, even the least competent orc will find no flashing pointer on the tower, figure out that their mission has failed, stop immediately, and wait for a rescue operation.

Saruman knows that his servants are not very brilliant minds, so he does not expect the delivery of the present to be quick, but he wants it to be successful at the very least. Therefore, it makes sense to assign to this task as competent an orc as possible; on the other hand, competent orcs are rare and pulling them out of their current activities may entirely disrupt those activities. Hence, given the description of Isengard's transport system, find the maximum number d such that there exists a way Saruman can put the flashing pointers, so that an orc with a level of incompetence equal to d can be assigned to deliver the present from the entrance tower and is guaranteed to carry out the order with success, reaching Orthanc.

Input

The first line contains two integers n and m — the number of towers and the number of one-directional passages between them ($2 \le n \le 4 \cdot 10^5$; $0 \le m \le 4 \cdot 10^5$). In the next m lines the descriptions of the passages follow. Each line contains three integers u_i, v_i, t_i — the numbers of the starting and the ending towers of a passage, and the number of seconds it takes for a loaded orc to travel along this passage ($1 \le u_i, v_i \le n$; $1 \le t_i \le 10^6$). There can be several passages between the same pair of towers, in any direction, as well as passages that lead from a tower to itself — in other words, loops, multiple passages, and symmetric pairs of passages are allowed.

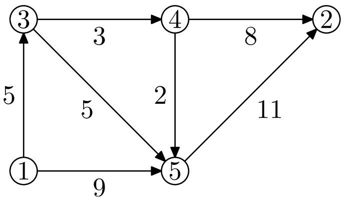
The entrance tower is numbered 1 and Orthanc is numbered n.

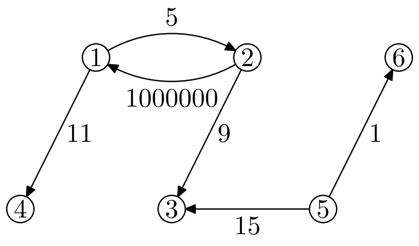
Output

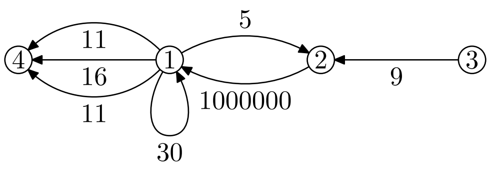
Print one integer d — the maximum incompetence of an orc who is guaranteed to complete the delivery. If with any level of incompetence it is possible, print the number n (the number of towers). On the other hand, if with any level of incompetence it is impossible, print -1.

Examples

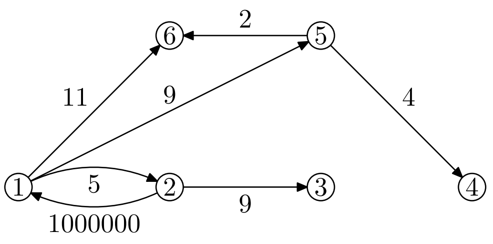
standard input	standard output	Illustration

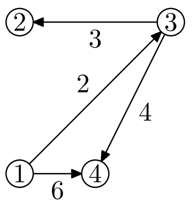
5 7
1 3 5
4 5 2
3 4 3
1 5 9
4 2 8
5 2 11
3 5 5
	2	

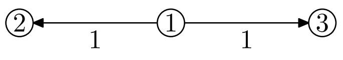
6 6
1 2 5
2 3 9
1 4 11
2 1 1000000
5 3 15
5 6 1
	-1	

4 7
1 2 5
1 1 30
3 2 9
1 4 11
1 4 16
2 1 1000000
1 4 11
	4	

2 0	-1	

6 7
1 2 5
2 3 9
1 6 11
2 1 1000000
1 5 9
5 6 2
5 4 4
	1	

4 4
1 4 6
1 3 2
3 2 3
3 4 4
	1	

3 2
1 2 1
1 3 1
	0	

Problem J. Judicious Watching

Time limit:	3 seconds
Memory limit:	1024 megabytes

Jill loves having good grades in university, so she never misses deadlines for her homework assignments. But even more, she loves watching the series and discussing it with her best friend Johnny. And unfortunately, today she needs to choose between these two activities!

Jill needs to complete n homework tasks. The i-th task would require a_i minutes to complete and needs to be submitted to the teacher at most d_i minutes from now. Also, there are m new episodes of the series that Johnny and Jill want to discuss. The j-th episode lasts l_j minutes. Jill can complete tasks in any order, but she needs to watch the episodes in the order they come. Neither completing a homework task nor watching an episode can be interrupted after starting.

Johnny and Jill need to agree on a time t_k when they would have a call to discuss the series. They are not sure yet which time to choose. For each possible time, compute the maximum number of episodes Jill could watch before that time while still being able to complete all n homework tasks in time.

Note that for the purpose of this problem we assume that discussing the series with Johnny at time t_k does not consume significant time from Jill and **can happen even if she is in the middle of completing any of her homework tasks**.

Input

There are several test cases in the input. The input begins with the number of test cases T ($1 \le T \le 1\,000$).

Each test case starts with a line with three integers n ($1 \le n \le 200\,000$) — the number of homework tasks, m ($1 \le m \le 200\,000$) — the number of episodes, and q ($1 \le q \le 200\,000$) — the number of possible times for the call with Jill.

The second line contains n integers a_i ($1 \le a_i \le 10^9$) — the number of minutes it takes to complete the task. The next line contains n integers d_i ($1 \le d_i \le 10^{15}$) — the deadline before which this task must be completed. The next line contains m integers l_j ($1 \le l_j \le 10^9$) — the length of episodes in the order they need to be watched. The next line contains q integers t_k ($1 \le t_k \le 10^{15}$) — the possible times of call with Jill.

It is possible to complete all tasks within their respective deadlines.

The sum of each of n, m, q over all test cases in input doesn't exceed $200\,000$.

Output

For each test case output a single line with q integers — for each possible time t_k the maximum number of episodes Jill can watch.

Example

standard input	standard output

2
1 2 3
10
15
5 5
5 15 20
3 4 5
8 100 8
10 150 20
2 32 1 1
9 200 51 50 10
	1 1 2
	1 4 2 2 1

Problem K. Knowns and Unknowns

Time limit:	3 seconds
Memory limit:	1024 megabytes

Two math professors have office hours on the same day. Students visit each professor to present their assignment solutions, one by one. For the whole semester, both professors established some fixed order of the students in which they should visit the professor. There are n students denoted with integers from 1 to n. Each professor's order of students is a permutation of numbers from 1 to n.

Today only some of the students visited the university; let A be the set of numbers that denote the students that were at the university today. All of the students from the set A have visited both professors, and all of the students outside of the set A haven't visited any professor.

Each of the professors made a list of the students they have talked with, in the same order the students have visited. Note that the list has to correspond to the order the professor has established, with the only difference that the students outside of the set A are missing in it. It is the beginning of the year, so the professors didn't have a chance to get to know every student. So for the students that a professor knows, the list contains their identifier, but for those that the professor doesn't know, the list contains -1.

Consider an example: the first professor's order is $[1, 2, 3, 4]$, and the second professor's — $[3, 2, 4, 1]$. The list made by the first professor today is $[1, -1, -1]$, and the list made by the second professor is $[3, -1, 1]$. Based on the lists, we can immediately see that three students have visited the university today. We can infer that the set A was either $\{1, 2, 3\}$ or $\{1, 3, 4\}$.

You are given two permutations — the orders established by each professor; you are also given two lists that professors made today. Your task is to help the professors. Based on the provided data, determine for each student whether they definitely visited the university, definitely did not, or whether this cannot be determined. Note that professors could have confused the students, so there is a possibility that the given data is inconsistent.

Input

The first line contains a single integer T ($T \geq 1$) — the number of test cases to solve.

Then the description of test cases follows.

The first line of the test case contains a single integer n — the number of students ($1 \leq n \leq 2000$).

The second line of the test case contains n distinct integers $p_{1,1}, p_{1,2}, \ldots, p_{1,n}$ — the order established by the first professor, meaning that student $p_{1,1}$ comes first, and $p_{1,n}$ comes last ($1 \leq p_{1,i} \leq n$).

The third line of the test case contains n distinct integers $p_{2,1}, p_{2,2}, \ldots, p_{2,n}$ — the order established by the second professor in the same format ($1 \leq p_{2,i} \leq n$).

The fourth line of the test case contains an integer k — the number of students that visited the university today ($1 \leq k \leq n$).

The fifth line of the test case contains k integers $s_{1,1}, s_{1,2}, \ldots, s_{1,k}$ — the first professor's list. Each student appears in the list at most once ($s_{1,i} = -1$ or $1 \leq s_{1,i} \leq n$).

The sixth line of the test case contains k integers $s_{2,1}, s_{2,2}, \ldots, s_{2,k}$ — the second professor's list in the same format ($s_{2,i} = -1$ or $1 \leq s_{2,i} \leq n$).

The total sum of n in all T test cases doesn't exceed 2000.

Output

For each test case, output a single string. If the given data is inconsistent, print a single word "`Inconsistent`". Otherwise, print a string consisting of n characters, the i-th of which is '`Y`' if the i-th student visited the university today, '`N`' if the i-th student didn't visit the university today, or '`?`' if it cannot be determined.

Examples

standard input	standard output

2
4
1 2 3 4
3 2 4 1
3
1 -1 -1
3 -1 1
4
1 2 3 4
3 2 4 1
3
1 -1 2
3 -1 1
	Y?Y?
	Inconsistent

2
3
1 2 3
2 1 3
2
-1 2
-1 -1
3
1 2 3
3 2 1
2
1 3
2 -1
	YYN
	Inconsistent

Problem L. Legacy Screensaver

Time limit:	3 seconds
Memory limit:	1024 megabytes

On a very old operating system, a screensaver consists of two rectangles flying around the screen. The screen is W pixels wide and H pixels high. Consider the origin to be in the top-left corner of the screen, the x-axis to go from the origin to the right, and the y-axis to go from the origin to the bottom.

Rectangle i ($i = 1, 2$) has a width of w_i pixels, a height of h_i pixels, initially its top-left corner has coordinates (x_i, y_i), and its initial movement direction is $(\delta x_i, \delta y_i)$, where each of δx_i and δy_i is either -1 or 1. At the end of each second, rectangle i's top-left corner coordinates instantly change by $(\delta x_i, \delta y_i)$.

Whenever rectangle i touches the left or the right border of the screen, the value of δx_i changes sign before the next second. Similarly, whenever rectangle i touches the top or the bottom border of the screen, the value of δy_i changes sign before the next second. Whenever rectangle i touches two borders of the screen at the same time (which can only happen at the corner of the screen), both δx_i and δy_i change sign.

As a result of the above, both rectangles stay fully within the screen at all times. Informally, collisions of the rectangles with the screen borders are perfectly elastic. Note, however, that rectangle movement is still discrete: each rectangle moves instantly by 1 pixel in both directions at the end of each second.

You are curious how often these two rectangles overlap. The rectangles are considered to be overlapping if their intersection has a positive area.

Let $f(t)$ be the number of integers $\tau = 0, 1, \ldots, t-1$ such that the rectangles overlap during second τ (where second 0 is before the rectangles start moving).

Find the limit of $\frac{f(t)}{t}$ as $t \to \infty$ as an irreducible fraction. It can be shown that this limit is a rational number.

Input

Each test contains multiple test cases. The first line contains the number of test cases T ($1 \le T \le 1000$). The description of the test cases follows.

The first line of each test case contains two integers W and H, denoting the width and the height of the screen ($3 \le W, H \le 4000$).

The next two lines describe the two rectangles. Each rectangle is described by six integers w_i, h_i, x_i, y_i, δx_i, δy_i, describing the i-th rectangle and denoting its width, its height, the coordinates of its top-left corner, and its initial movement direction ($1 \le w_i \le W - 2$; $1 \le h_i \le H - 2$; $0 < x_i < W - w_i$; $0 < y_i < H - h_i$; $\delta x_i, \delta y_i \in \{-1, 1\}$).

The sum of the values of $W + H$ across all test cases is guaranteed to not exceed 8000.

Output

For each test case, print a non-negative integer p and a positive integer q, separated by a slash ('/') without spaces, meaning that the limit of $\frac{f(t)}{t}$ as $t \to \infty$ is equal to $\frac{p}{q}$. The fraction must be irreducible — that is, the greatest common divisor of p and q must be equal to 1.

Example

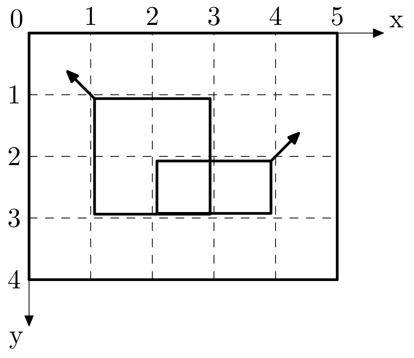
standard input	standard output

2
3 3
1 1 1 1 1 1
1 1 1 1 1 -1
5 4
2 2 1 1 -1 -1
2 1 2 2 1 -1
	1/2
	1/3

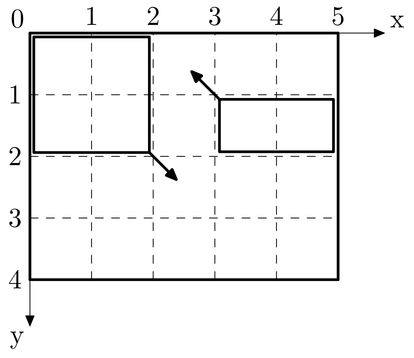
Note

For the second test case, the state of rectangles during the first few seconds is shown in the following pictures. The rectangles overlap during seconds $\tau = 0$ and $\tau = 6$. Thus, for example, $f(8) = 2$.

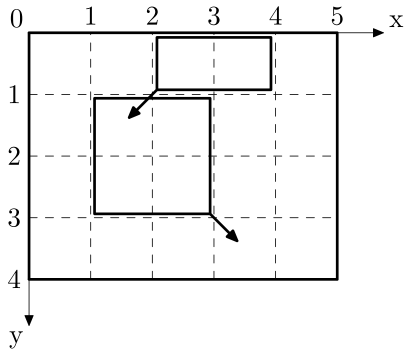
$\tau = 0$



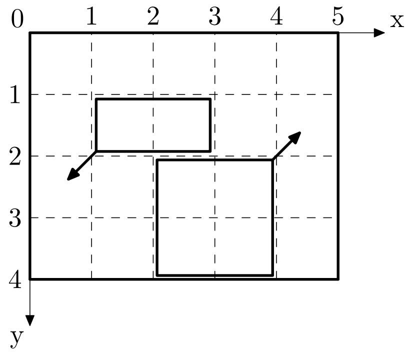
$\tau = 1$



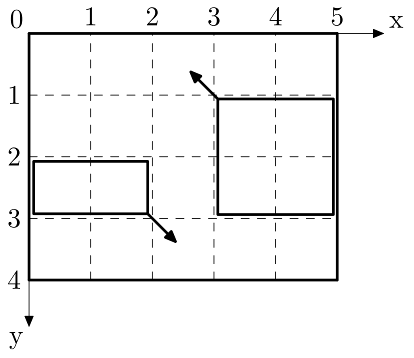
$\tau = 2$



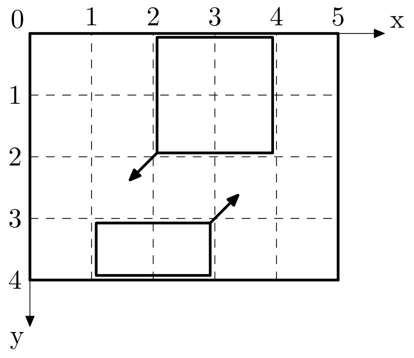
$\tau = 3$



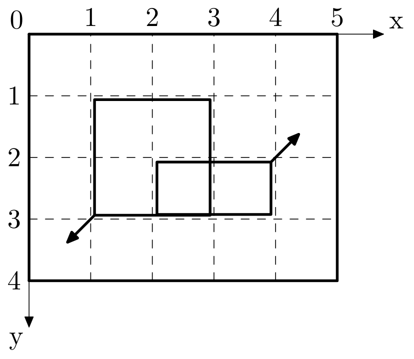
$\tau = 4$



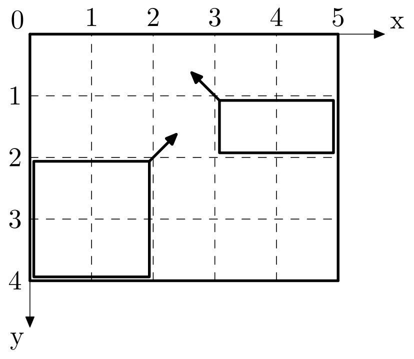
$\tau = 5$



$\tau = 6$



$\tau = 7$



Problem M. Managing Cluster

Time limit:	3 seconds
Memory limit:	1024 megabytes

You want to write a cluster manager extension that will improve your product performance. Your product has n services (numbered from 1 to n) and is hosted on a cluster with $2n$ machines (numbered from 1 to $2n$). Each service is running in exactly two replicas. Each replica is run on some machine. Each machine runs exactly one replica of some service.

One of the key factors of this cluster's performance is the network. Some pairs of machines are connected directly and can transfer data between them very efficiently. There are exactly $2n - 1$ direct connections, and it is possible to transfer data between any two machines using direct connections. In other words, direct connections form a tree.

During the deployment, all $2n$ replicas were assigned to machines. Your extension gets the direct connections list and the sequence $a_1, a_2, \ldots, a_{2n}$, where a_i is the number of the service that will be running on machine i. Your extension can swap some replicas between machines. The swap operation takes two machines i, j and swaps values a_i and a_j. Each machine is allowed to participate in at most one swap operation. Your extension should make some swap operations that maximize the cluster performance.

Due to the fact that most data will be transferred between two replicas of the same service, the cluster performance is measured as the number of services that have two replicas running on machines connected directly. Help to write the extension that will maximize the cluster performance.

Input

The first line contains a single integer T ($1 \le T \le 10^5$) — the number of test cases. Descriptions of test cases follow.

The first line of each test case contains a single integer n ($1 \le n \le 10^5$).

The second line contains $2n$ integers $a_1, a_2, \ldots, a_{2n}$ ($1 \le a_i \le n$). It is guaranteed that each value from 1 to n appears exactly twice in this sequence.

Each of the next $2n-1$ lines contains two integers u and v ($1 \le u, v \le 2n$, $u \ne v$), meaning that machines u and v are connected directly. Direct connections are guaranteed to form a tree.

It is guaranteed that the sum of n for all test cases does not exceed 10^5.

Output

For each test case on the first line print a single integer k ($0 \le k \le n$) — the number of swap operations the extension wants to make.

Each of the next k lines should contain two integers i, j ($1 \le i, j \le 2n$, $i \ne j$) — swap operations. Each number from 1 to $2n$ should appear at most once.

Note that the order of operations is not important. After applying swap operations, the cluster performance should be the maximum possible. You can print any answer that satisfies the requirements.

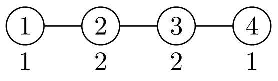
Example

standard input	standard output

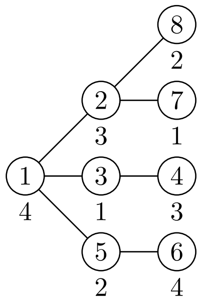
3
2
1 2 2 1
1 2
2 3
3 4
4
4 3 1 3 2 4 1 2
1 2
3 1
3 4
5 1
5 6
2 7
2 8
3
1 1 2 2 3 3
1 2
1 3
1 4
1 5
1 6
	1
	1 3
	3
	1 5
	8 3
	4 7
	0

Note

In the first test case only replicas of service 2 run on directly connected machines, so the performance is 1. The performance can be increased to 2 by swapping replicas between machines 1 and 3.



In the second test case no two replicas run on directly connected machines, so the performance is zero. The performance can be increased to 3 by performing swaps $1 - 5$, $8 - 3$, and $4 - 7$ so that replicas of services 2, 3, and 4 run on directly connected machines. It can be shown that it is impossible to get performance 4 here.



In the third test case only replicas of service 1 run on directly connected machines, so the performance is 1. It is obvious that here the performance cannot be made any bigger.